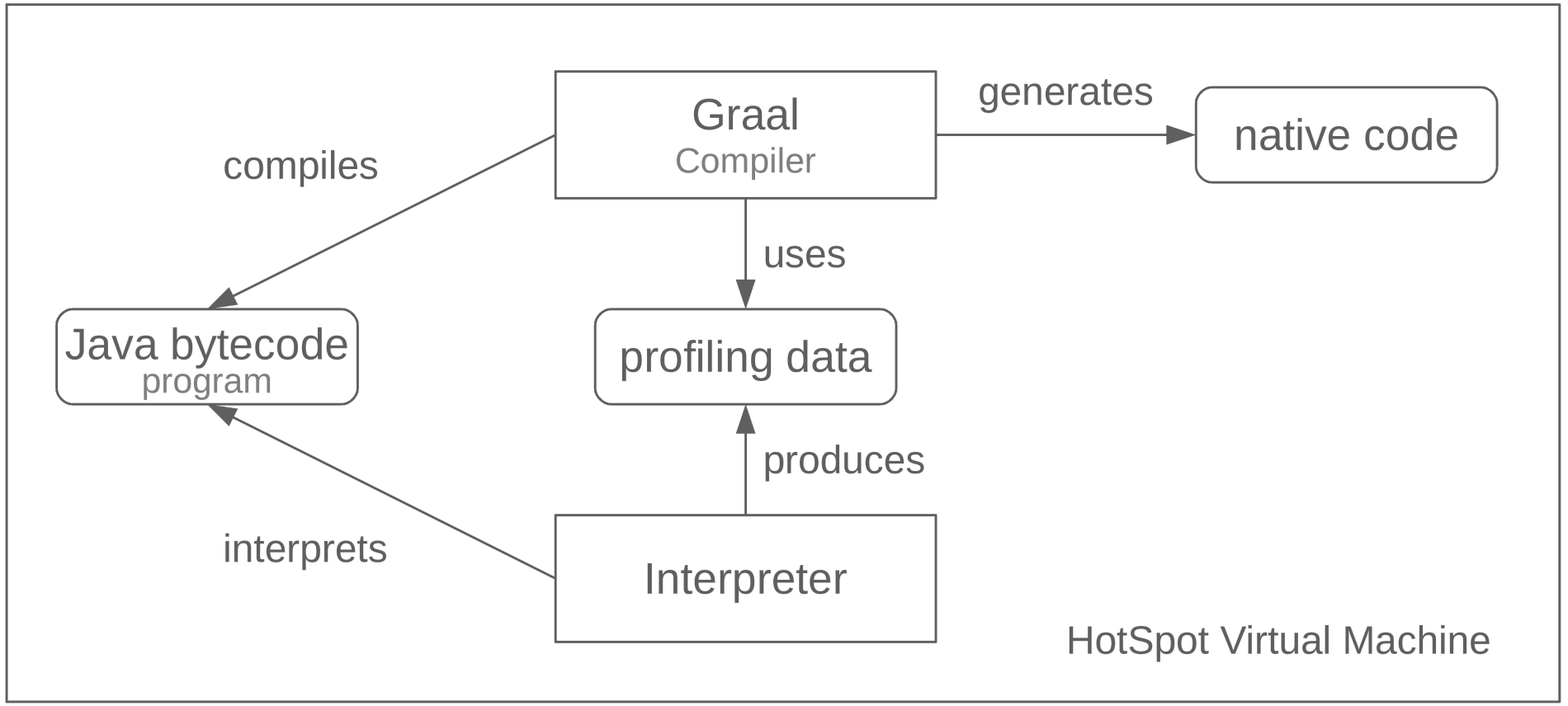
Speculative optimizations in the Graal Just-In-Time compiler

Gilles Duboscq
Oracle Labs
VMSS — June 2, 2016

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Context



Profiling

- Branches with counts (if, switches, …)
- Types with counts (virtual calls, checkcasts, …)
- Exceptions at calls

Motivating Examples

```
if (unlikely()) {
  // complex
} else {
  // simple
}
```

Motivating Examples

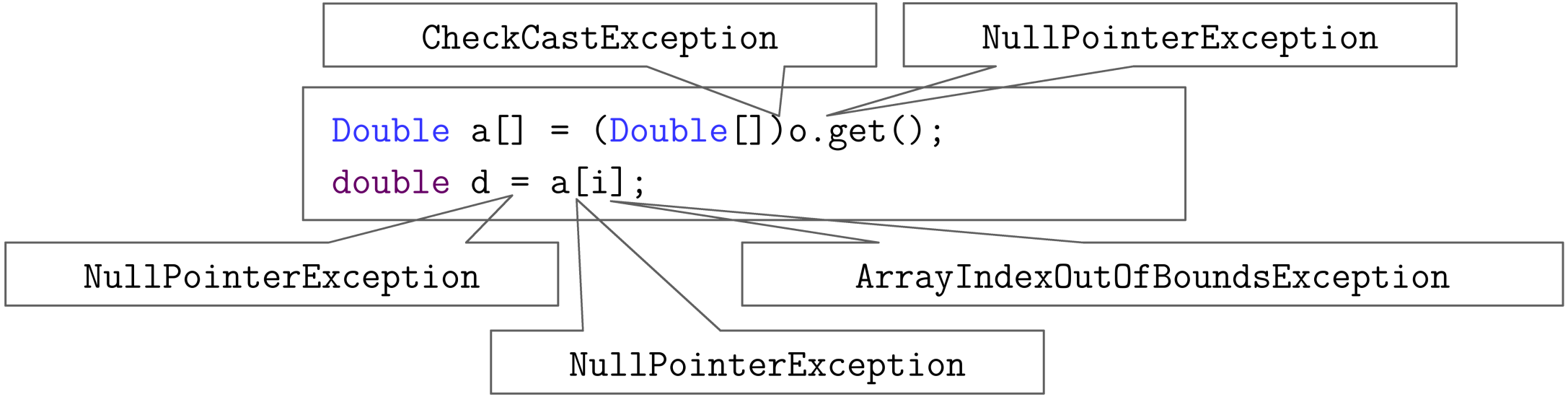
```
if (a instanceof Foo) {
  // ...
}
```

Motivating Examples

```
Foo f = getFoo();
if(f.get()) {
  // complex
} else {
  // simple
}
```

```
class Foo {
  public boolean get()
{
    return false;
  }
}
```

Motivating Examples



Speculative Optimizations

An optimization is **speculative** if it relies hypotheses that are not verified during compilation.

Guards

Guards check conditions during execution and **deoptimize** on failure.

Deoptimization exits the compiled code and resumes execution in the interpreter.

Assumptions

Assumptions are speculative invariants maintained by the VM.

When invalidated, the VM pauses execution and **deoptimizes** all code that relied on this assumption.

Revisiting the Examples

```
if (unlikely()) {
  // complex
} else {
  // simple
}
```

```
if (unlikely()) {
  deoptimize;
}
// simple
```

Revisiting the Examples

```
if (unlikely()) {
  // complex
  f = new Foo();
} else {
  // simple
  f = new Bar();
}
```

```
if (unlikely()) {
  deoptimize;
}
// simple
f = new Bar();
```

Revisiting the Examples

```
if (a.getClass() == Foo.class) {
  // ...
}
```

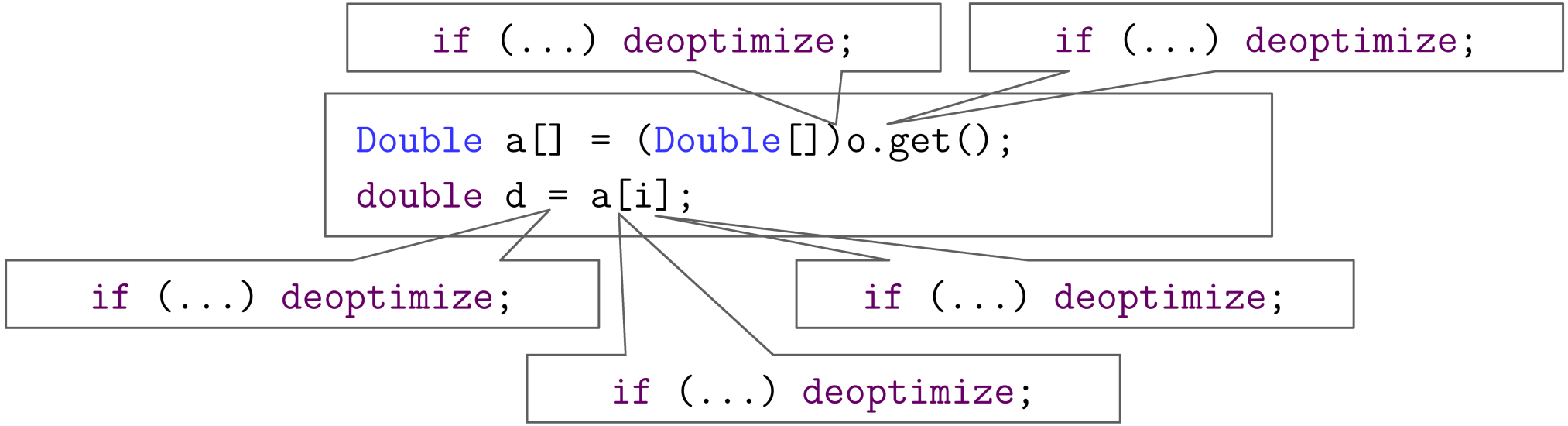
Revisiting the Examples

```
Foo f = getFoo();
if(f.get()) {
  // complex
} else {
  // simple
}
```

```
Foo f = get();
if(f.getClass() != Bar.class) {
  deoptimize;
}
// simple
```

```
class Bar {
  public boolean get()
{
    return false;
  }
}
```

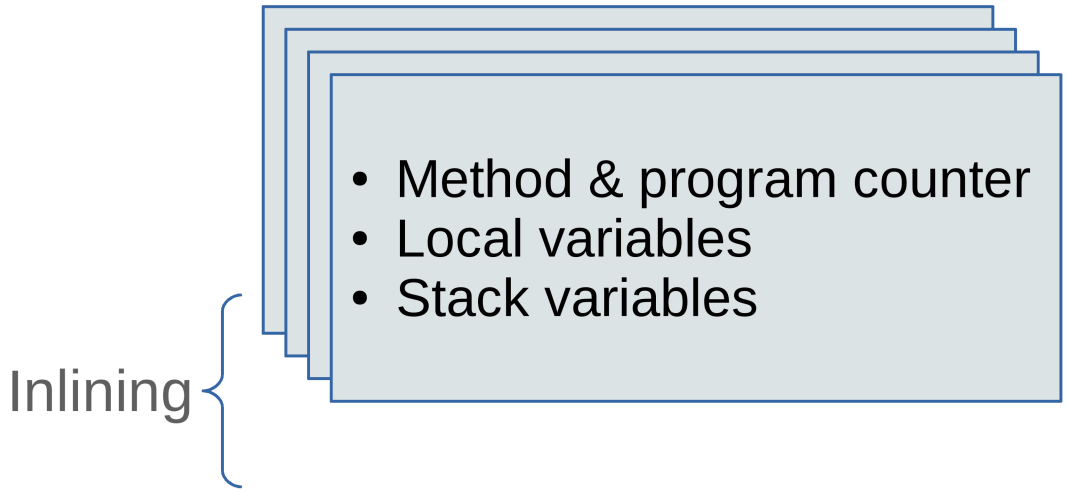
Revisiting the Examples



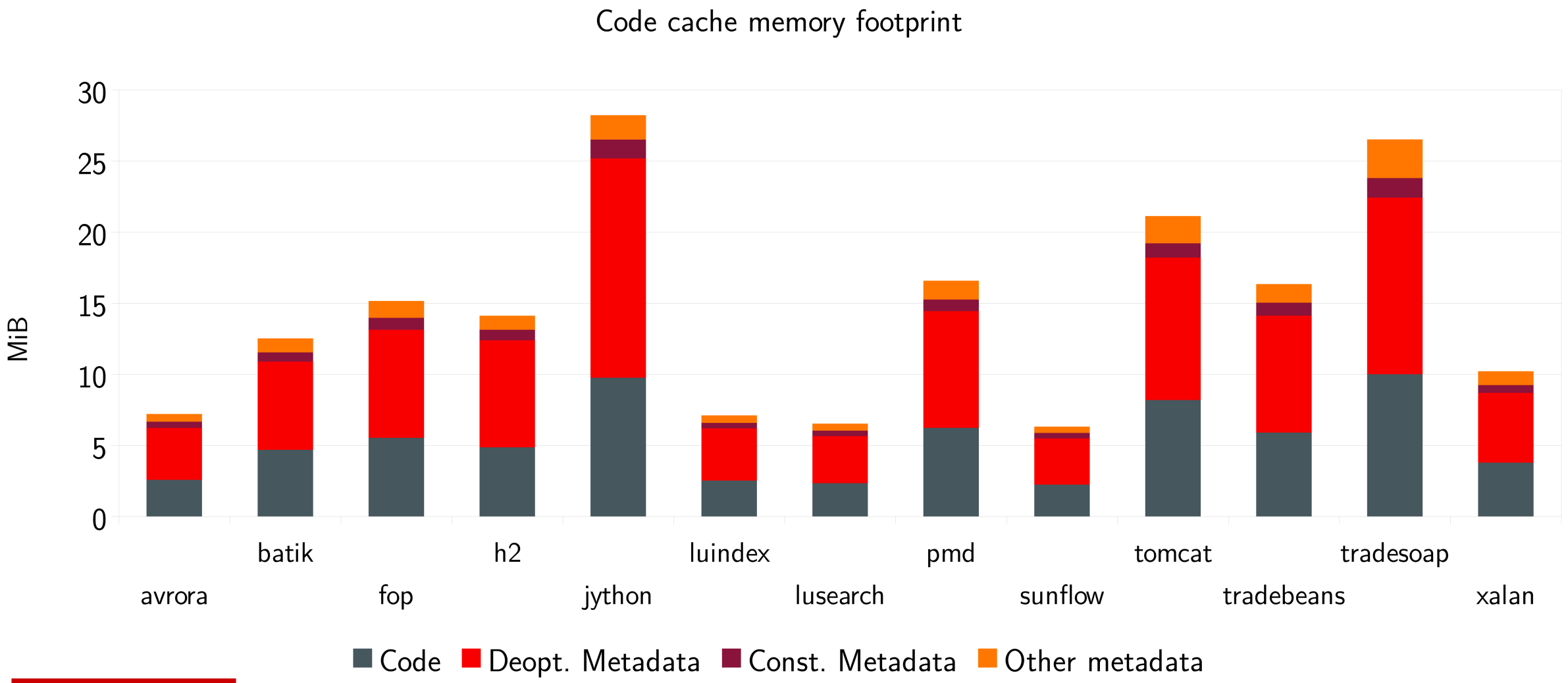
Speculation: Run-time Costs

Guards are **run-time** checks executed by the compiled code

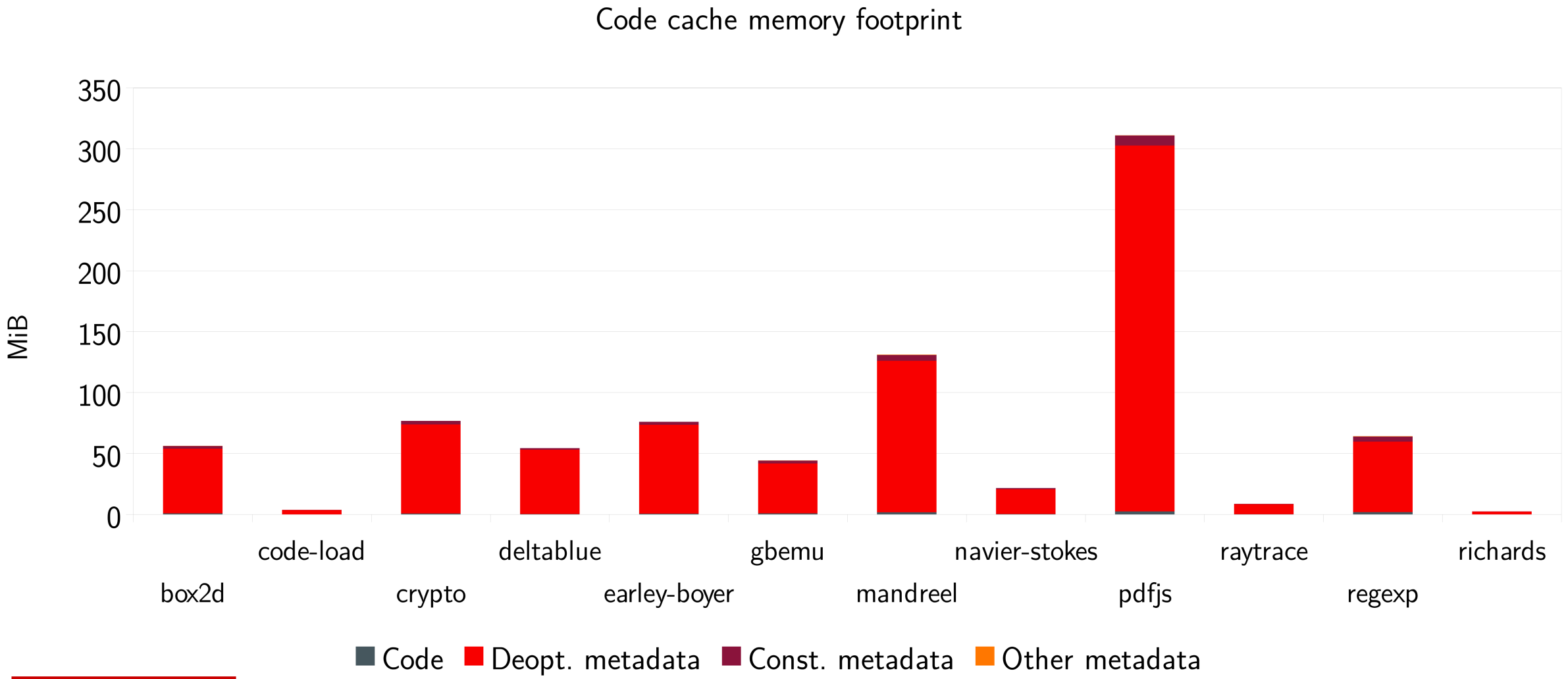
Speculation: Memory Costs



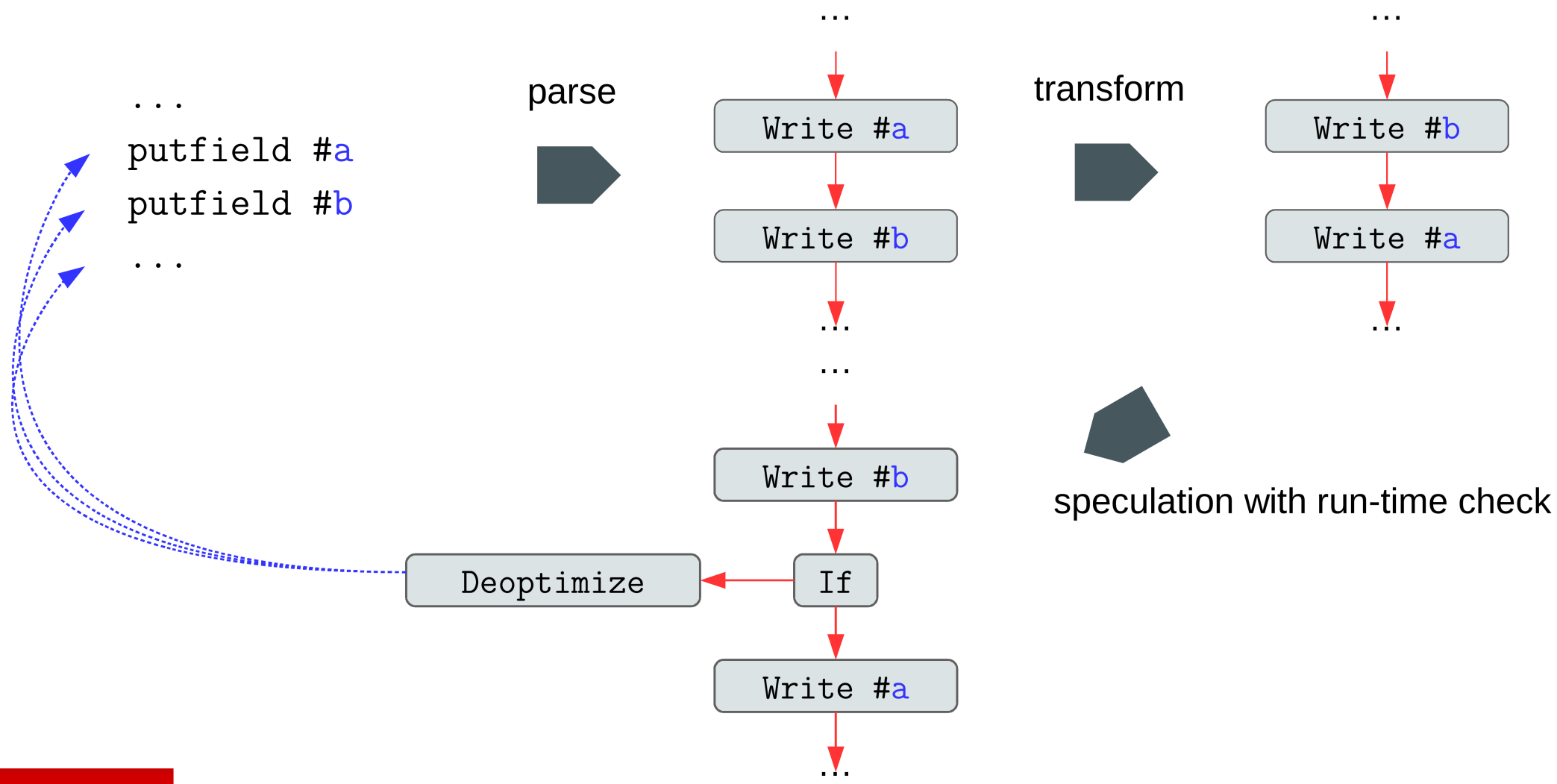
Speculation: Memory Costs



Speculation: Memory Costs



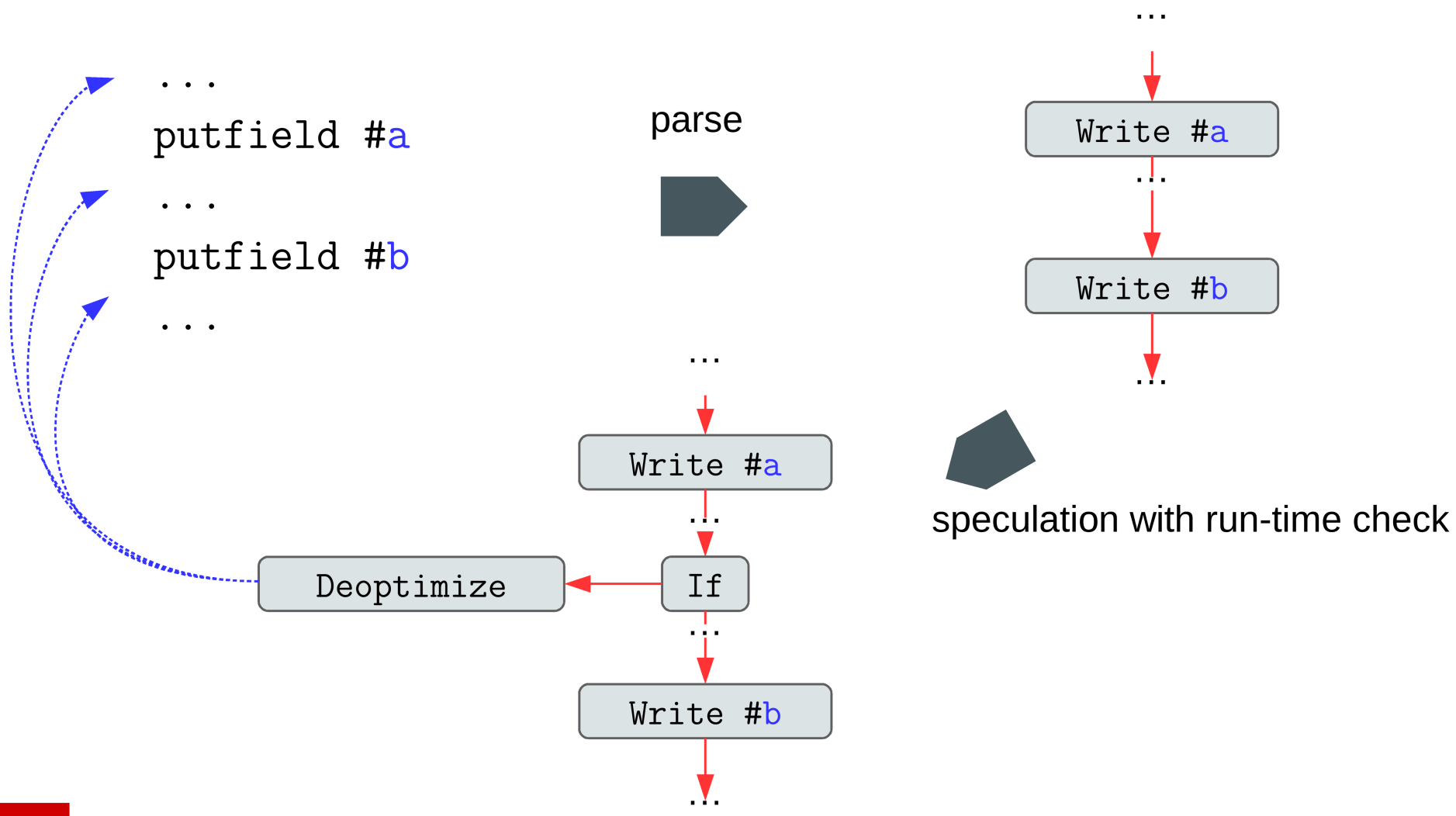
Deoptimization Targets



Deoptimization Targets

Two side-effecting instruction can only be reordered if there is no deoptimizing instruction between them.

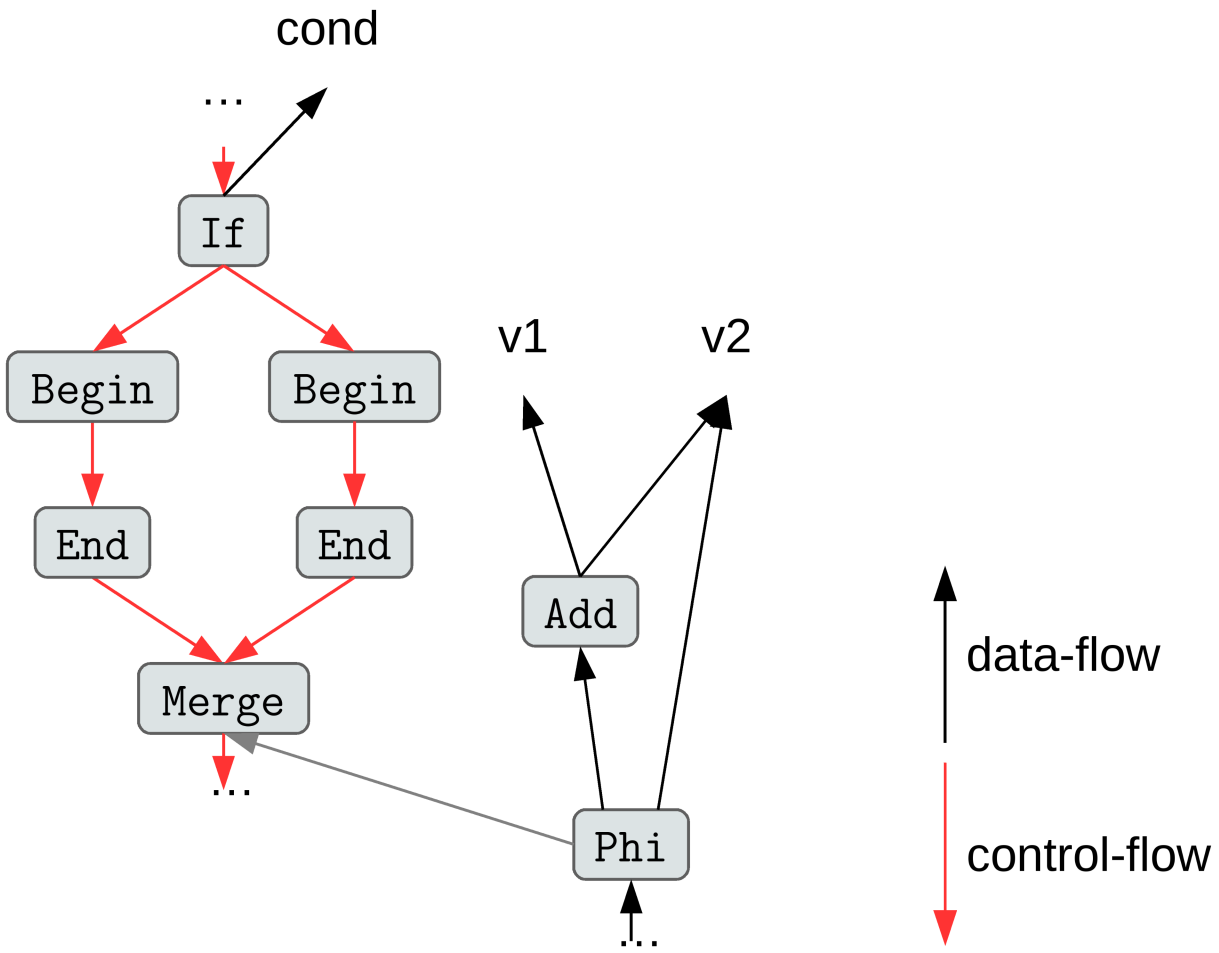
Deoptimization Targets



Deoptimization Targets

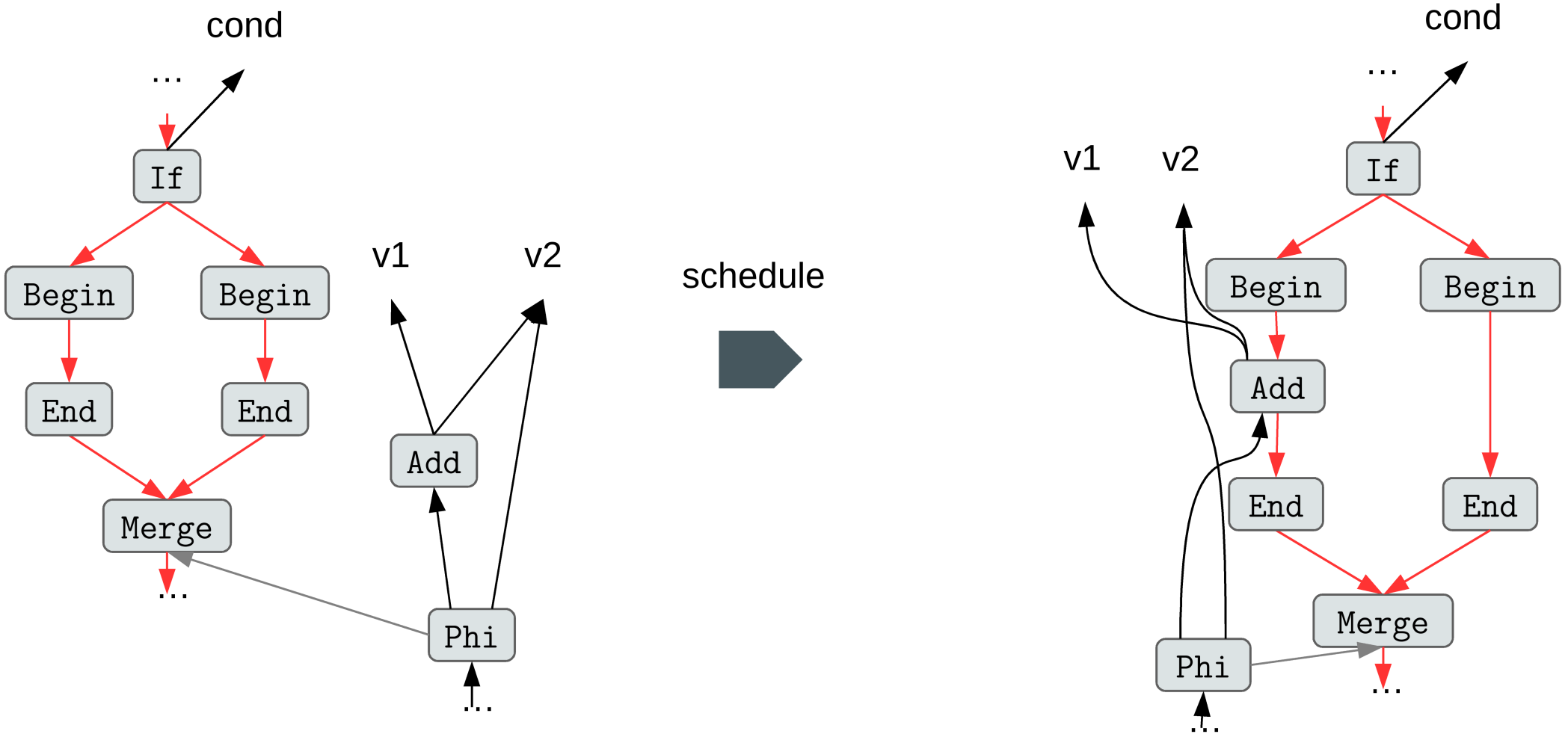
If there is a deoptimizing instruction between two side-effecting instructions, the deoptimization target must be between the corresponding side-effecting bytecodes.

Intermediate Representation

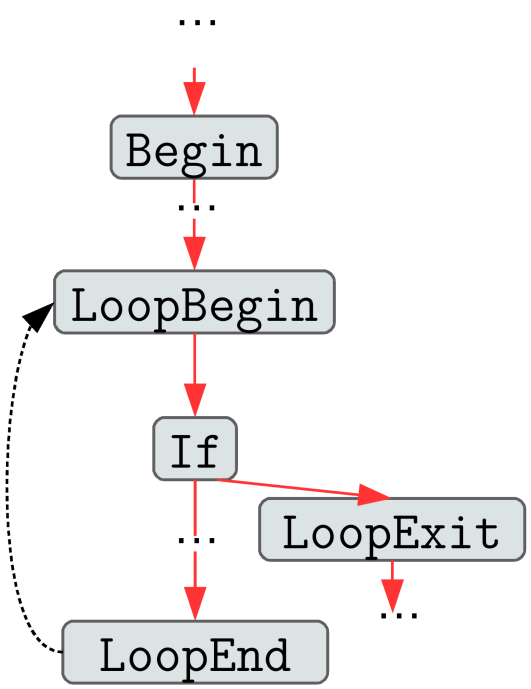



```
if (cond) {
  foo = v1 + v2;
} else {
  foo = v2;
}
```

Intermediate Representation

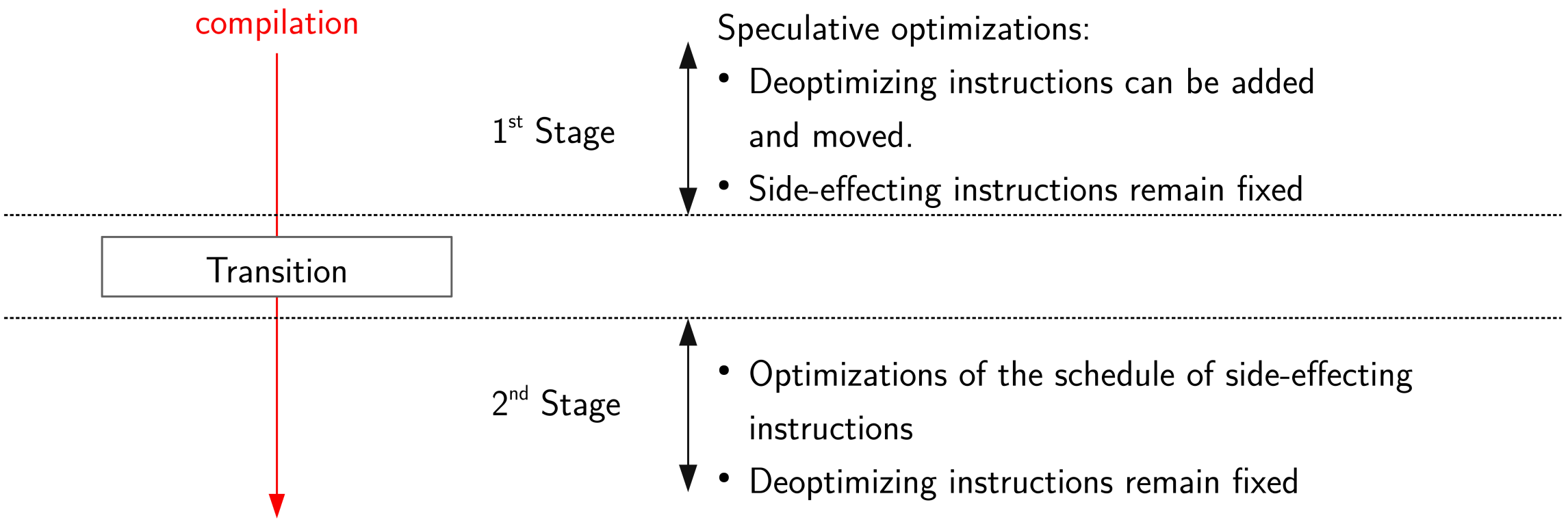


Intermediate Representation

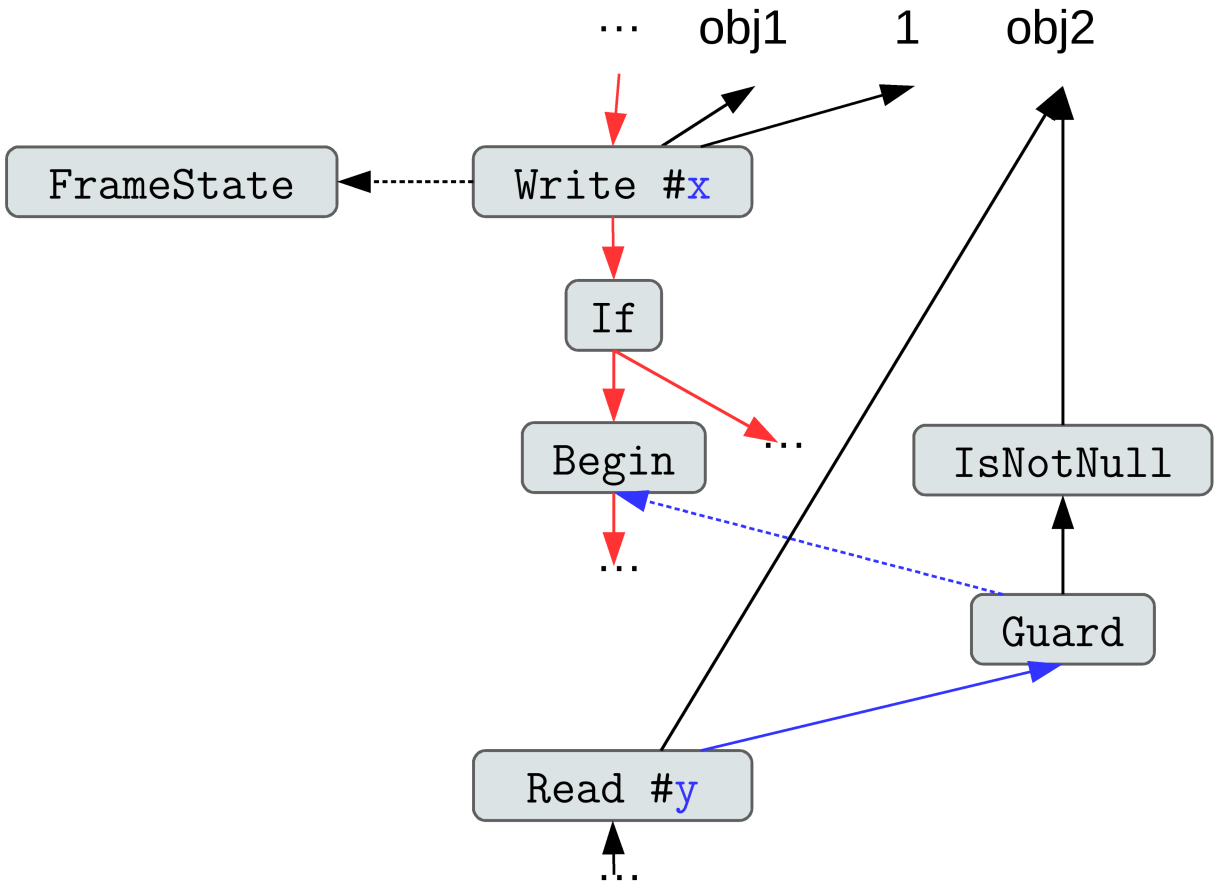


- No irreducible control-flow
- Explicit loop structure

Optimization Stages

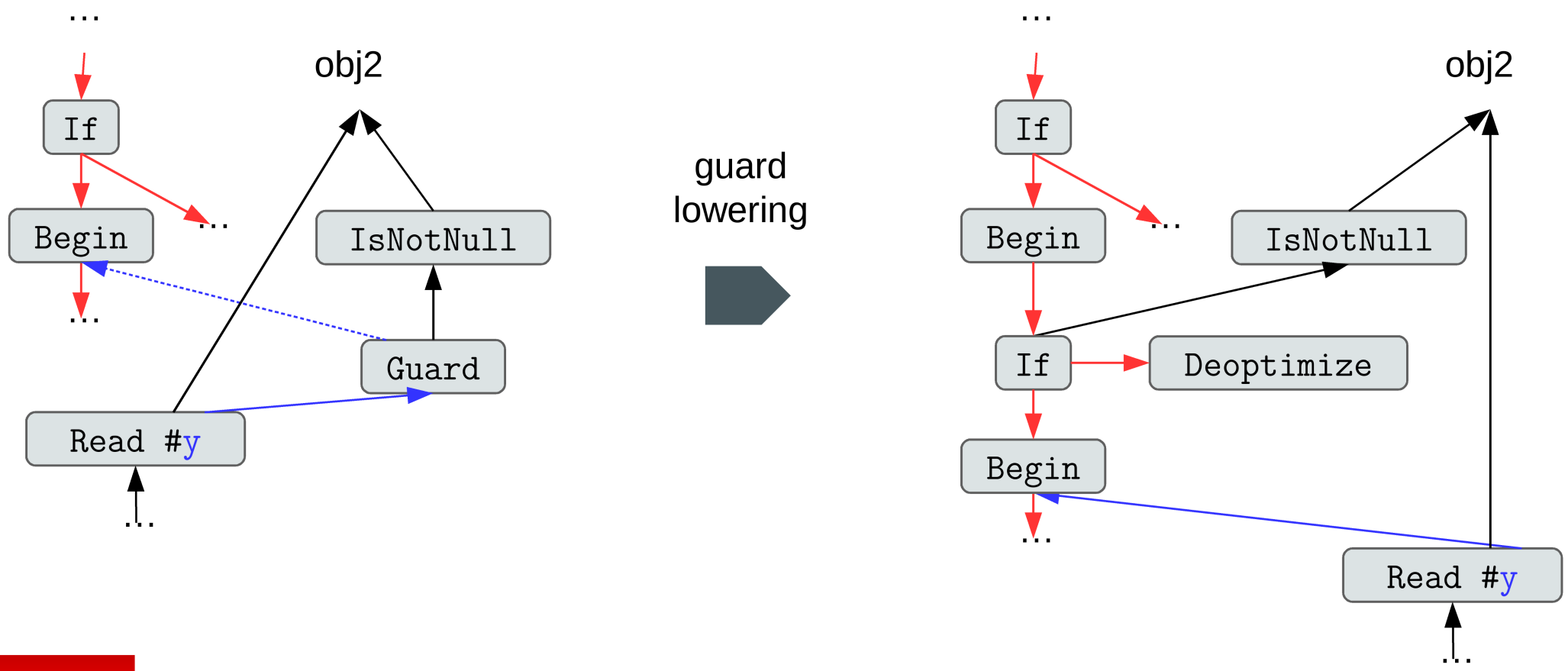


Optimization Stages: 1st Stage

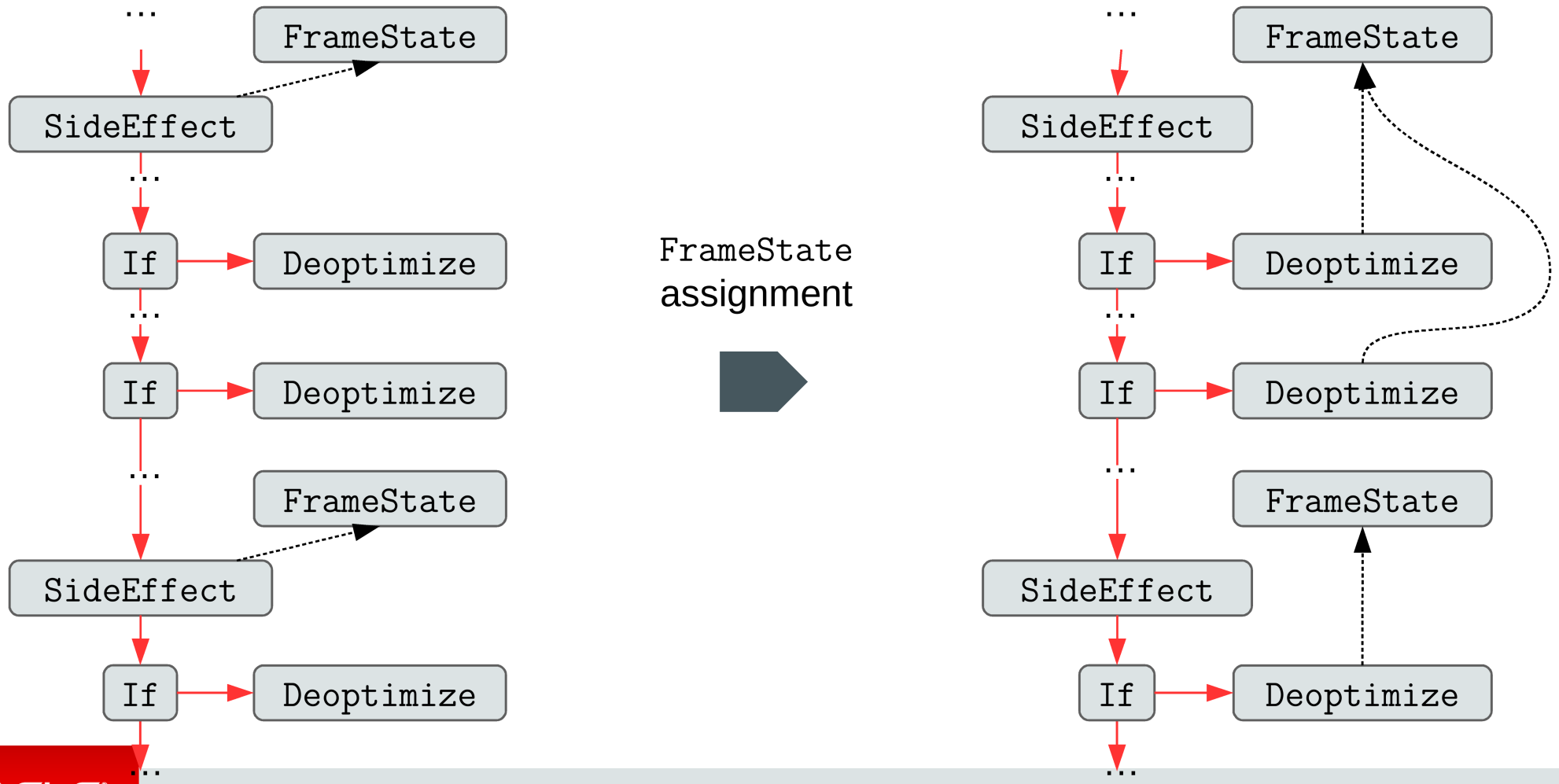



```
...
obj1.x = 1;
if (c) {
  t = obj2.y;
} else {
...
}
...
```

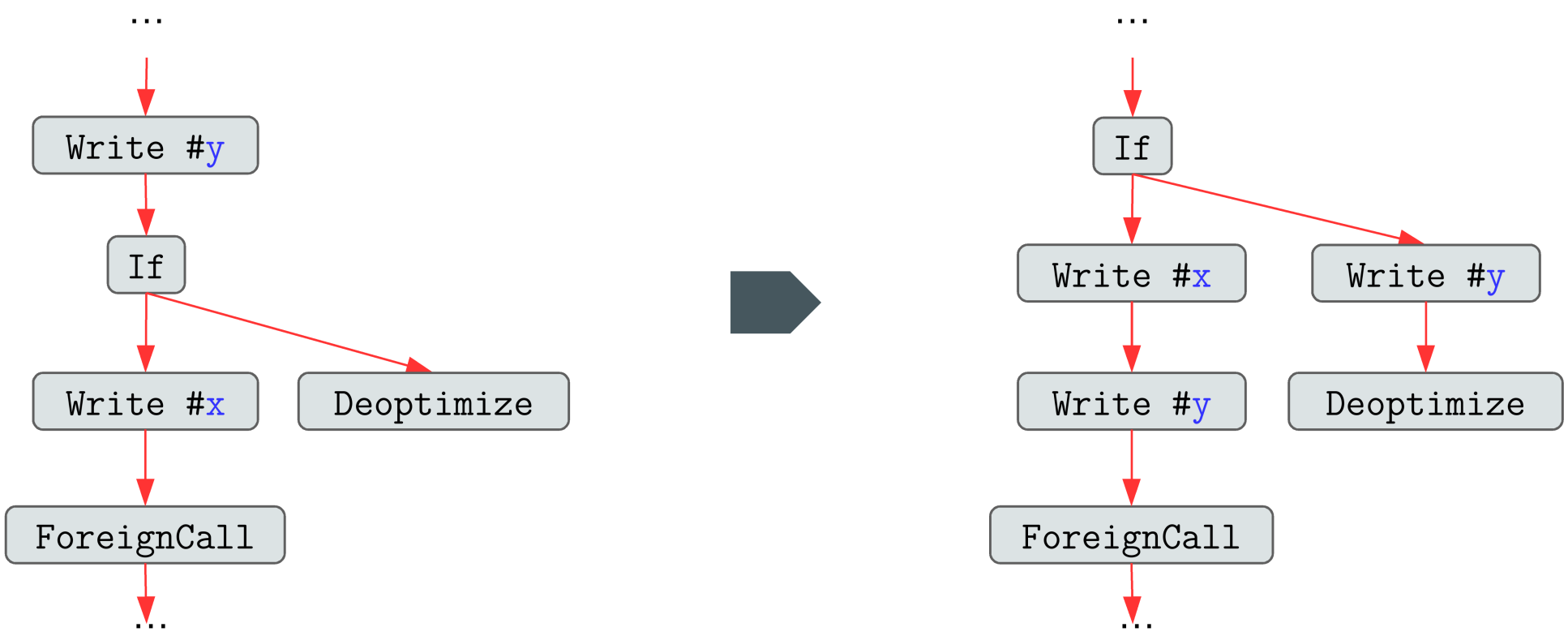
Optimization Stages: Transition



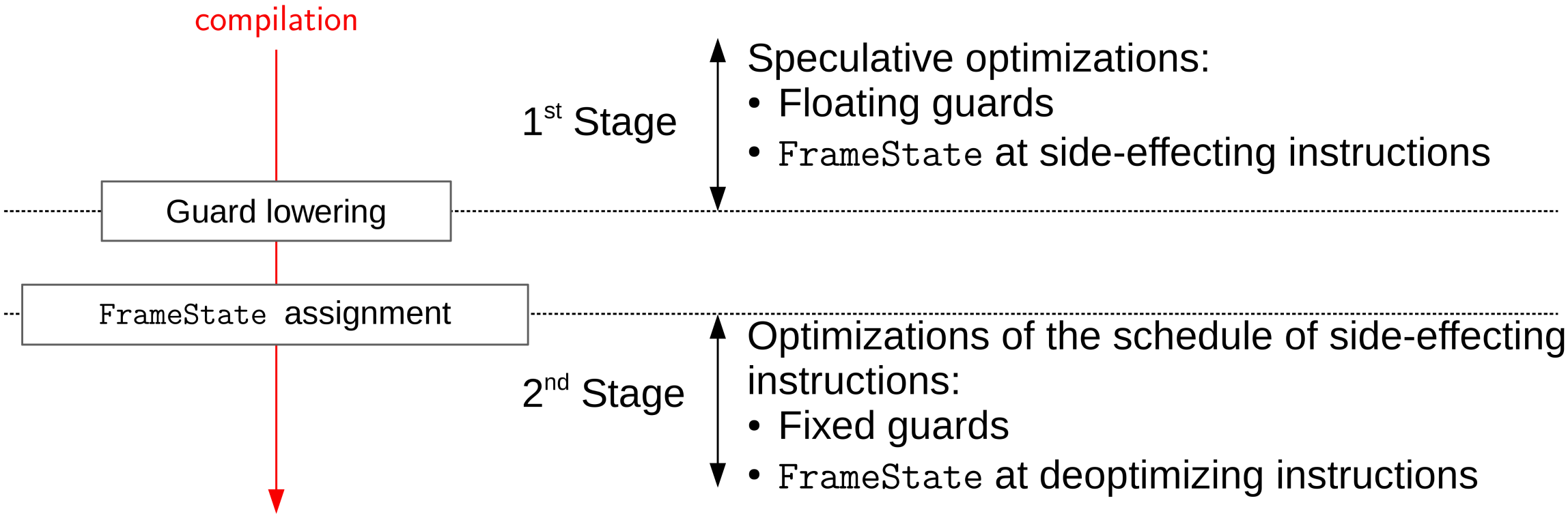
Optimization Stages: Transition



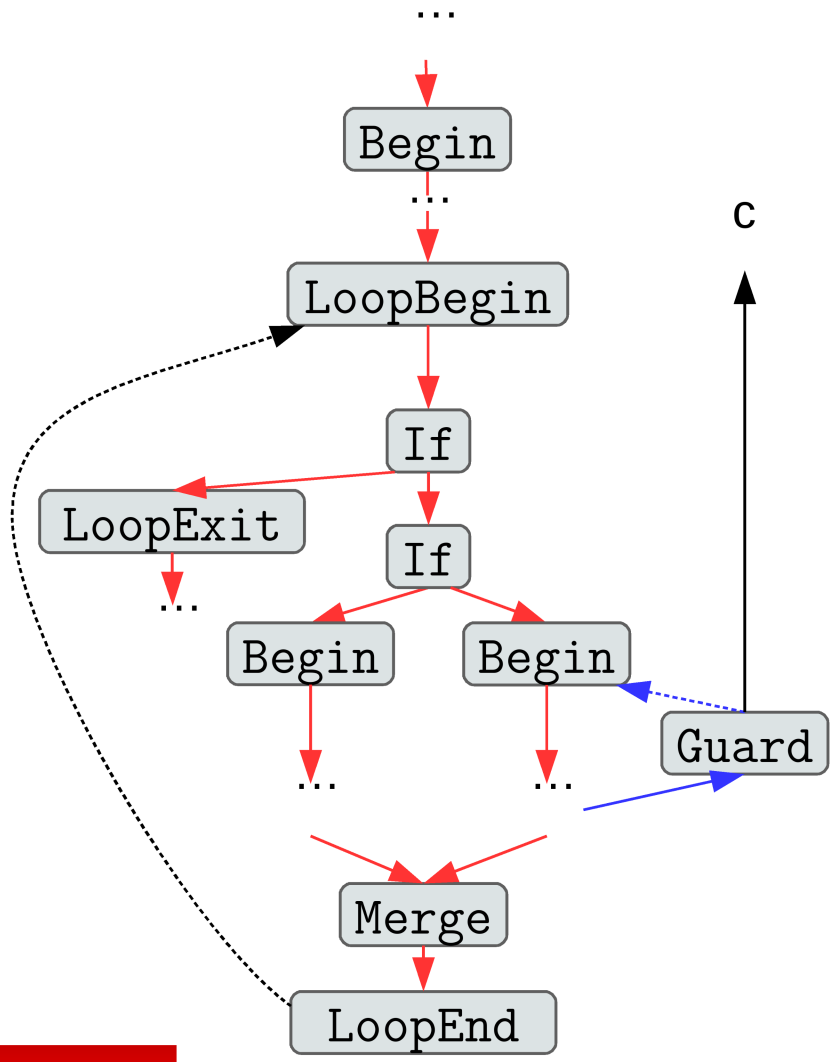
Optimization Stages: 2nd Stage



Optimization Stages

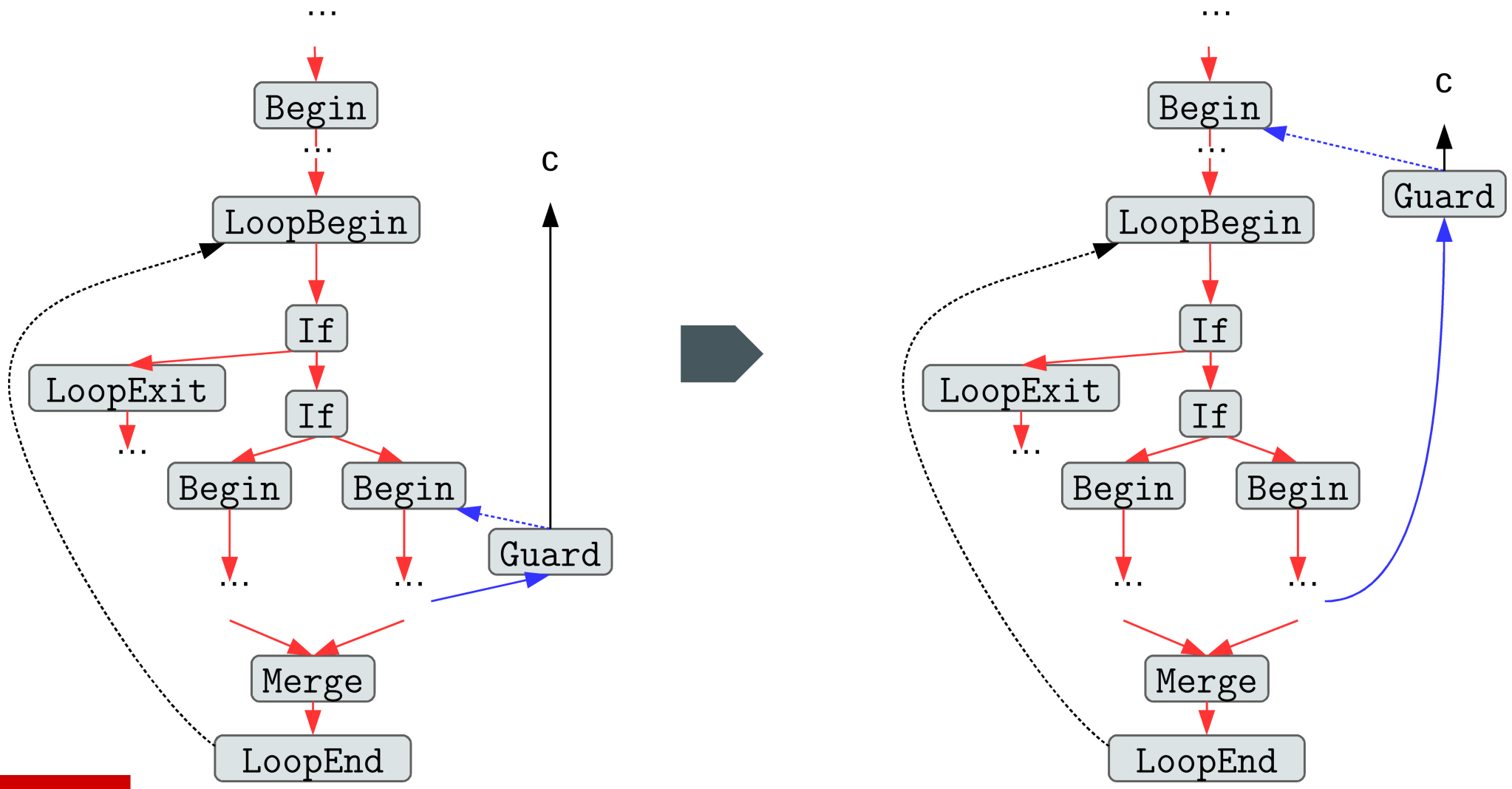


Speculative Guard Motion




```
boolean c = ...
for (...) {
  if (...) {
    guard(c);
  }
}
```

Speculative Guard Motion



Speculative Guard Motion

- Refinements:
  - Rewrite induction variable comparisons

    i < c → min(i) < c && max(i) < c
  - Processing order

    Hoisting a guard may make new code loop invariant and allow other guards to be hoisted
  - Check relative frequencies

Speculative Guard Motion

- Refinements:
  - Rewrite induction variable comparisons

    i < c → min(i) < c && max(i) < c
  - Processing order

    Hoisting a guard may make new code loop invariant and allow other guards to be hoisted
  - Check relative frequencies

Speculative Guard Motion

```
for (int i = 0; i < n; i++) {
    x = …; y = …;
    if (x > 100) {
        guard(a != null);
        guard(i < a.length);
        y -= a[i];
    }
}
```

```
guard(a != null);
for (int i = 0; i < n; i++) {
    x = …; y = …;
    if (x > 100) {
        guard(i < a.length);
        y -= a[i];
    }
}
```

Speculative Guard Motion

```
for (int i = 0; i < n; i++) {
    x = …; y = …;
    if (x > 100) {
        guard(a != null);
        guard(i < a.length);
        y -= a[i];
    }
}
```

```
guard(a != null);
for (int i = 0; i < n; i++) {
    x = …; y = …;
    if (x > 100) {
        guard(n <= a.length);
        y -= a[i];
    }
}
```

Speculative Guard Motion

```
for (int i = 0; i < n; i++) {
    x = …; y = …;
    if (x > 100) {
        guard(a != null);
        guard(i < a.length);
        y -= a[i];
    }
}
```

```
guard(a != null);
guard(n <= a.length);
for (int i = 0; i < n; i++) {
    x = …; y = …;
    if (x > 100) {
        y -= a[i];
    }
}
```

Alias analysis

```
for (int i = i0; i < a.length; ++i) {
  c[i] = a[i-1] * 2 + 1;
}
```

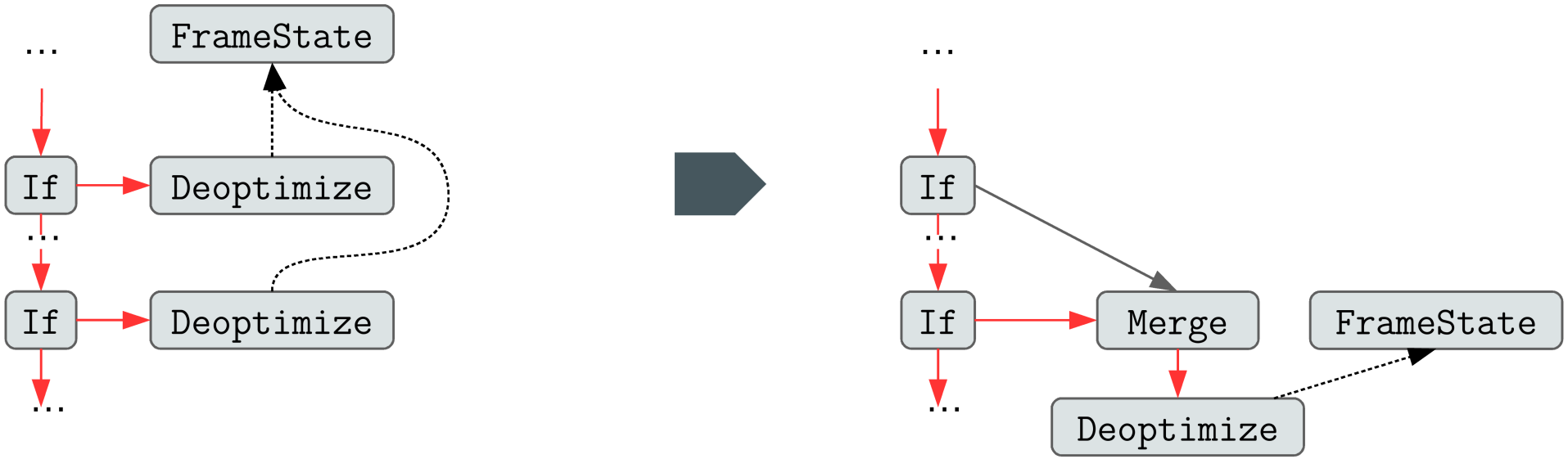
```
guard(c != a);
for (int i = i0; i < a.length; ++i) {
  c[i] = a[i-1] * 2 + 1;
}
```

Safepoint Elimination

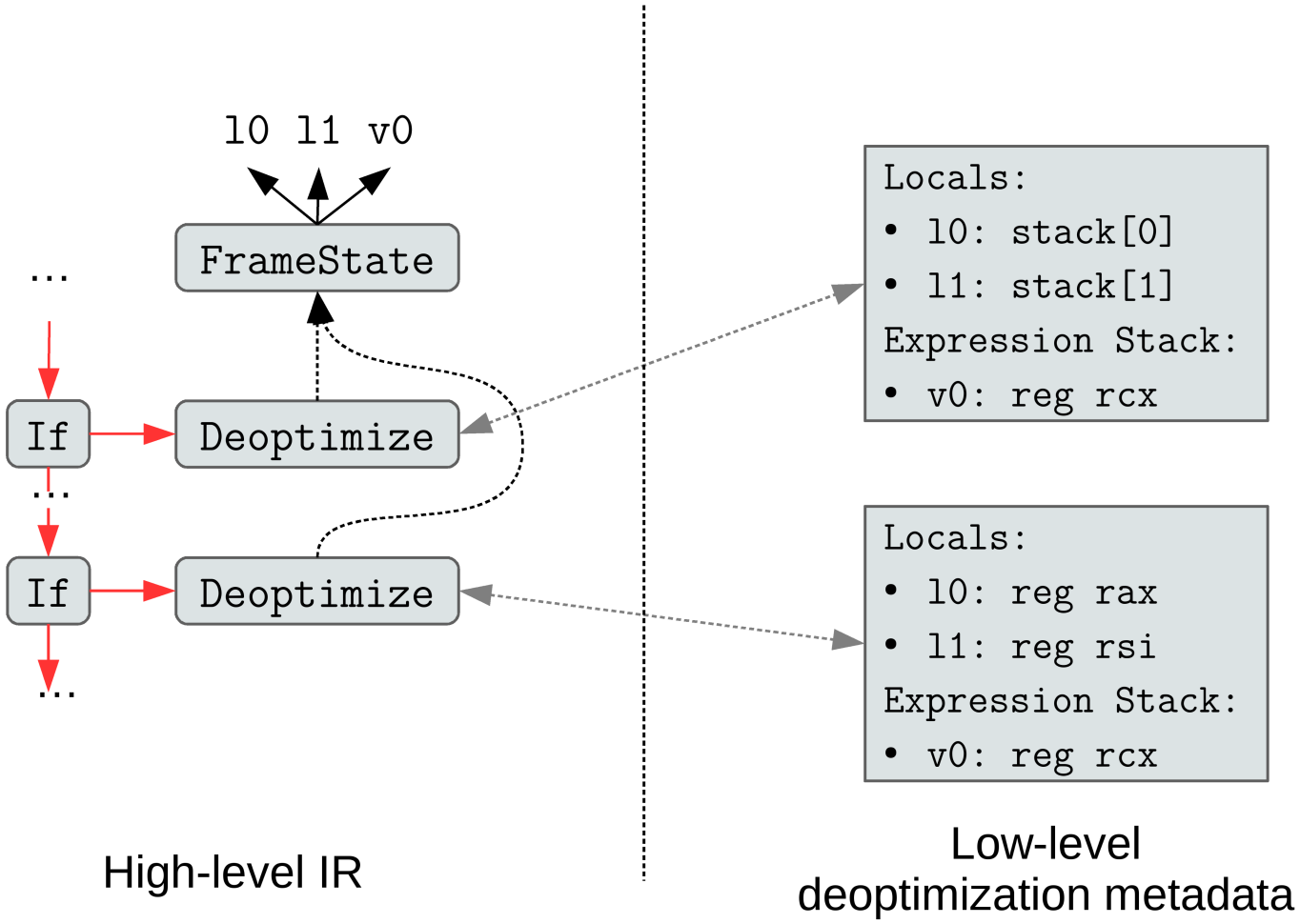
```
for (int i = 0; i < n; i+=2) {
  …
  safepoint();
}
```

```
guard(n < MAX_INTEGER);
for (int i = 0; i < n; i+=2) {
  …
}
```

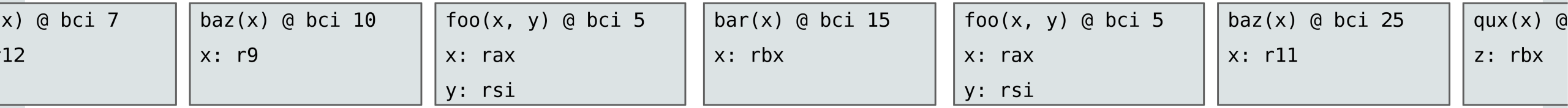
Deoptimization Grouping



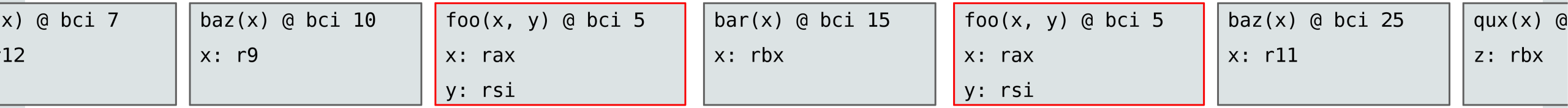
Deoptimization Grouping



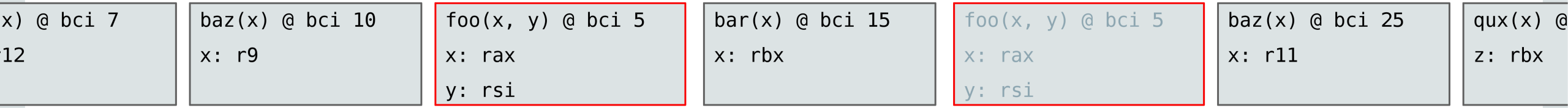
Debug Info Sharing



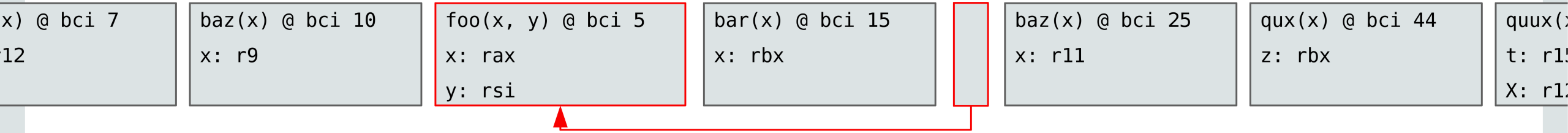
Debug Info Sharing



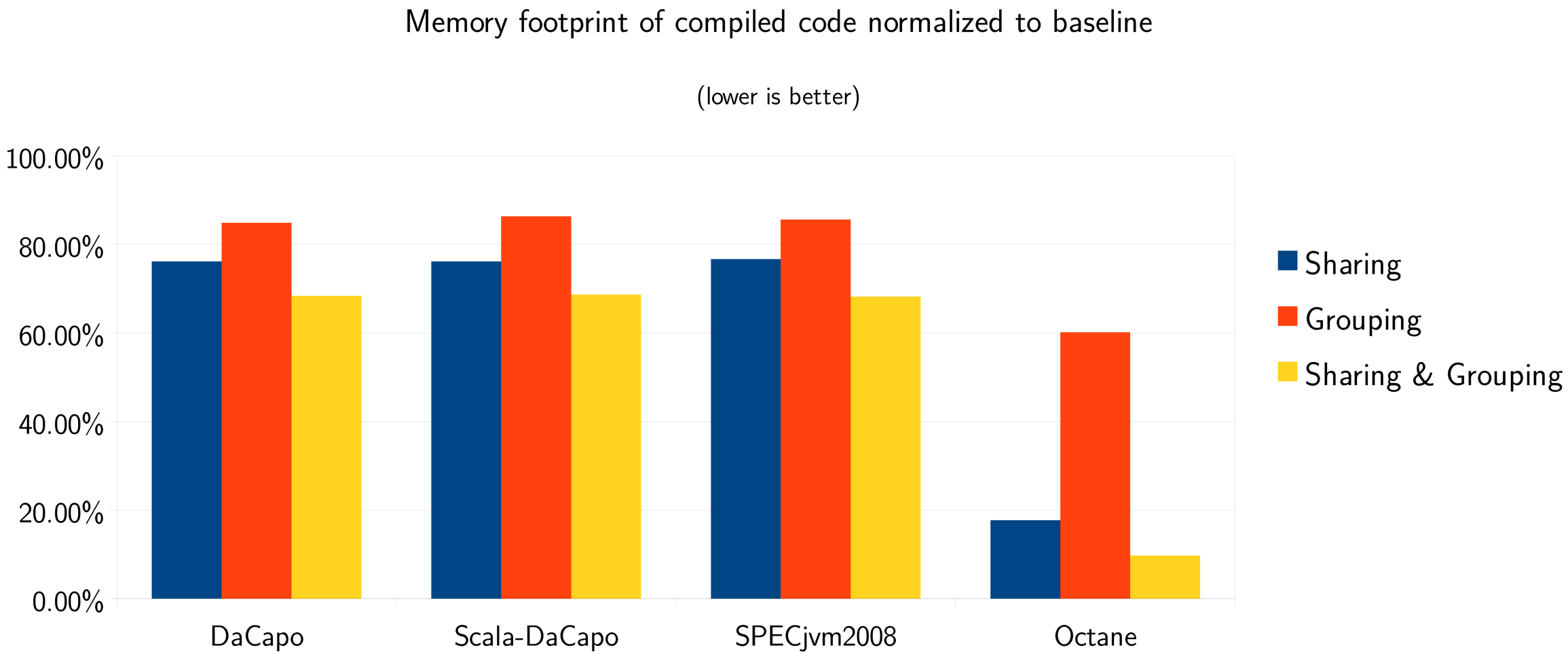
Debug Info Sharing



Debug Info Sharing



Deoptimization Grouping



Loop Fusion

```
double[] b = new double[a.length];
for (int i = 0; i < a.length; ++i) {
  b[i] = a[i] * 2;
}
double[] c = new double[a.length];
for (int i = 0; i < a.length; ++i) {
  c[i] = b[i] + 1;
}
```

```
double[] c = new double[a.length];
for (int i = 0; i < a.length; ++i) {
  c[i] = a[i] * 2 + 1;
}
```

Effect Sinking

```
for (int i = i0; i < n; ++i) {
  this.x = this.x * 2 + 1;
}
```

```
int tx = this.x;
for (int i = i0; i < n; ++i) {
  tx = tx * 2 + 1;
}
this.x = tx;
```

Questions


ORACLE®